Case Study of Using Soft Systems Methodology and UML in a Systems Design Problem within an Agile Environment

James Donohue[1] and Huseyin Dogan[2]

[1] British Broadcasting Corporation, London, United Kingdom
james.donohue@bbc.co.uk
[2] Bournemouth University, Bournemouth, United Kingdom
hdogan@bournemouth.ac.uk

Abstract. A generalised software system for the creation and delivery of web syndication feeds, based on the requirements of the British Broadcasting Corporation (BBC) World Service, is considered as an industrial case study in systems design within Agile environments. The design problem is considered first using perspectives from Soft Systems Methodology, leading to the production of a number of UML 2.0 artefacts through the process of analysing and designing the system, incorporating use case and class analysis and sequence and activity diagrams. The authors find that a number of features of SSM and UML remain applicable to information systems design, and reflect on the challenges of using these approaches in organisations such as the BBC that have embraced an Agile approach to software development. Finally suggestions are made for future work to further evaluate and draw connections between theory and current trends in industrial practice.

Keywords: UML, Soft Systems, Agile

1 Introduction

The British Broadcasting Corporation (BBC) World Service website portfolio, which currently includes 27 sites but will soon increase to 35, offers news to millions of users around the world in their own language. A publicly-stated ambition of the BBC [1] is to increase its global audience to 500 million by 2022. One way it can contribute to this effort is through web syndication and the provision of public 'feeds' of its content.

At the time of writing the existing World Service sites are being migrated to a new content management system (CMS) and the 'Public Feeds Adaptor' that previously provided web syndication services for these sites is going to be decommissioned. Therefore a new system based around the new CMS must be implemented in order to maintain web syndication capability for the existing 27 sites, and to ensure that it is available for the new ones that will be added.

This paper is an industrial case study of how a large media organisation such as the BBC has put different systems design methodologies into practice in developing a solution to this problem, considering the relevance of different aspects

of the approaches and the challenges of employing them within an industrial context.

1.1 Web syndication

Web syndication feeds allow website owners to publish information about new or changed content in a way that can be reused by other websites or end users. Standards for web syndication have existed in some form since 1997 [2], the dominant formats today being RSS (Really Simple Syndication) [3] and Atom [4]. Both of these are XML-based and offer a similar feature set, and for the purpose of this study they are treated as equivalent [5].

Syndication feeds are especially significant to a publisher such as the BBC World Service because they offer an alternate path for distributing news content where conventional website channels are ineffective or inaccessible. Unlike conventional web pages, RSS and Atom feeds can be downloaded and redistributed relatively easily, and even used as a data source for mobile apps, bots and other services. Users of RSS feeds (referred to as 'subscribers') typically use a generic 3rd-party 'reader' application installed on their computer or smartphone, that periodically makes an HTTP request to the BBC's feeds server to retrieve metadata about new and recently updated website content. Typical feed item metadata includes the titles of news articles and a short description, along with a link to the article itself [6].

In addition to this base level of information, based on editorial policies the BBC may choose to offer 'full text' feeds for certain audiences. These augmented feeds include the full body text of the article as well as its metadata (title, URL, etc.). This feature is required because users living in countries with state restrictions on Internet access may still be able to access RSS feeds by other means even where access to the associated website has been blocked. In other words, 'full text' feeds may be used to facilitate the circumvention of government-mandated censorship of external news sources, since such feeds can be circulated via e-mail, on a USB storage device or even via a custom app, in order to bypass such restrictions.

Beyond the well-established RSS and Atom formats, newer formats for syndicating web content, often proprietary in nature and using JSON (JavaScript Object Notation) syntax, are emerging. These formats open a range of new possibilties for sharing news content across different content platforms and devices, often with richer and more structured metadata. Although there is no specific requirement to support these formats at present, one requirement for the new syndication platform is to offer greater flexibility around output formats so that they could be added at a later stage. Notwithstanding this, recent work [7] has demonstrated a possible new role for the existing RSS and Atom formats as vehicles for the syndication of Semantic Web data.

2 Methodology

This study will examine how Soft Systems Methodology (SSM) and the Unified Modeling Language (UML) might be used in the analysis and design of a replacement system for web syndication, and to evaluate the usefulness of these approaches for the current problem. For the purposes of this study, we assume that a new 'ideal' system will be built that entirely replaces, and if possible improves upon, the previous one. However, other options are available, such as adapting the legacy system to interoperate with the new CMS. One reason for this assumption is to allow the authors freedom to use full range of UML features without being constrained by the existing architecture.

The rationale for the use of SSM and UML together is to determine whether they provide a useful and complementary set of approaches for systems design problems such as the one under examination here. The UML artefacts generated during the investigation will be used as the basis for the iterative design and development of the new system, using an Agile-based development process which is currently the *de facto* standard at the BBC.

Once the system is operational there will be further opportunities to reflect on the relevance of the modelling techniques used by comparing this study with the finished product. In addition, the authors' aim is that the results of this investigation, along with the areas for further research and analysis suggested in the final section, will provide insights as to how organisations such as the BBC might make use of these modelling techniques in future projects.

In the terminology of Soft Systems Methodology (SSM), the starting point for any effort such as this one is a perceived 'problematical situation' which is made the subject of a series of learning-focused activities. Checkland and Poulter [8] see SSM as a shift from the supposed 'positivism' of 1950s and 1960s systems engineering methods which sought to solve real-world problems objectively to a phenomenological approach in which the the *Weltanschauung*, or worldview, of all actors is emphasised. This is the focus of the next section.

3 Results

3.1 Contextualisation of the problem space through soft systems

The first Soft Systems activity centres around finding out about the problematical situation. To do this, a *rich picture* may be created as an aid to exploring the context of the problem, and particularly to capture the "rich moving pageant of relationships" [9] between the entities involved. Fig. 1 shows shows a rich picture for the present problem.

This rich picture suggests some of the actors whose worldview we should consider. Indeed, SSM's role analysis (known formally as 'Analysis One') expands the definition of 'issue owner' to be anyone who may be "concerned about or affected by the situation and the outcome" [8]. From this perspective, the owners of the issue(s) include the News Product Development Group, the BBC Trust, the Public Feeds Adaptor team and the World Service editorial staff, but also

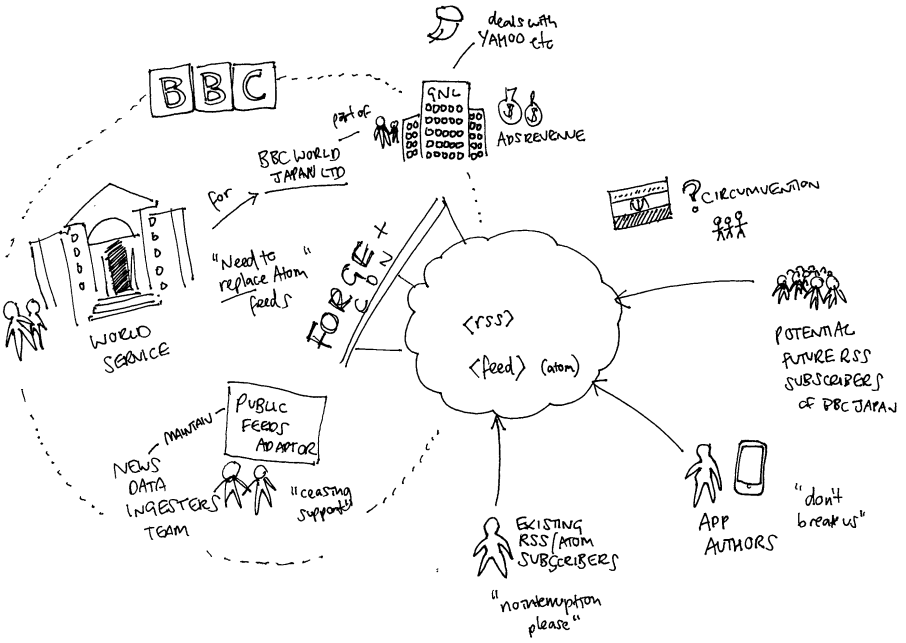

Fig. 1. Rich Picture of the BBC's problematical situation

members of the public currently subscribed to feeds, builders of third-party apps that consume the feeds, and even potential future users who do not currently use the feeds.

Next, SSM proposes the creation of a number of models showing 'purposeful activity' within a worldview that is made explicit. The starting point for each model is a root definition that describes the system as a statement using the PQR formula [8]. Taking one example from the rich picture, we can write the following root definition:

"A system to provide a new web syndication platform for World Service sites by automatically generating and serving up-to-date RSS feeds, in order to allow users to receive updates about new and changed content and therefore contribute to an increase in website traffic."

As a further step towards enriching our understanding of this system, Soft Systems encourages us to consider the mnemonic CATWOE [8]. Through this we might see a transformation (the T) from a deprecated legacy system with no support for the new CMS, to a better-supported and more flexible platform that will meet the BBC's wider needs in years to come. CATWOE offers a distinction between actors (who are involved in the system's activities), owners and customers who are affected by it. This allows us to refine our list of users given earlier.

Finally, we should give consideration to the environment (the E of CATWOE). There are several dimensions to this, including the operational environment of the BBC itself, but also the context of the changing landscape of web syndication in the age of Twitter and Facebook, with an apparent shift in the use of feed formats such as RSS. These considerations, and what impact they have on the actors involved, will be touched upon in the use case diagrams and other artefacts produced in the next section.

3.2 UML artefacts

In this section we present a series of UML [10] artefacts that explore the problem domain described above from various angles. These diagrams are organised according to some of the workflows within the software development lifecycle recommended by the Unified Process (UP).

Use case diagram From the root definition and list of actors given by soft systems, we can create the UML use case diagram shown in Fig. 2, which is a non-comprehensive depiction of some of the use cases that the system must support. For the purposes of this diagram four actors have been identified, which represent the key roles that users of the system may occupy. The naming of the roles favours typical organisational usage where this does not agree with modelling best practice. For example the Journalist role could have been named ContentEditor (since the use cases assigned to the role relate to editing content, and this role could be performed by a variety of different individuals not all of whom have the job title Journalist), but this may cause confusion for non-technical stakeholders because of the domain-specific significance of 'Editor'.

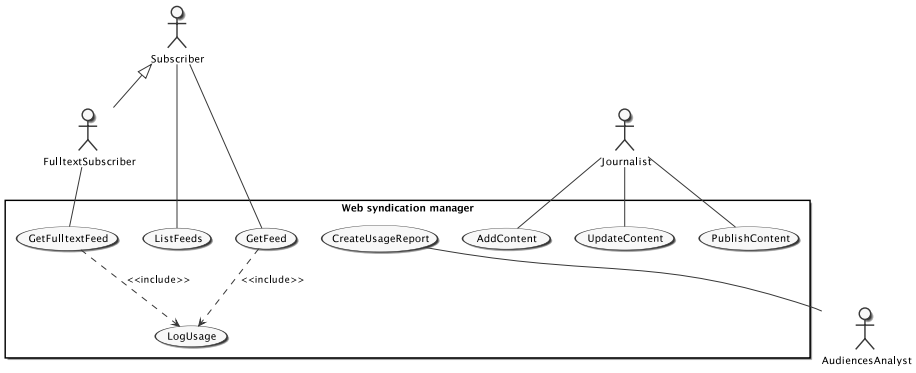

Fig. 2. A use case diagram

To include the requirement for 'full-text' feeds, the FulltextSubscriber actor is depicted as a specialised form of the Subscriber actor and therefore a generalisation relationship exists between the two. This entails that FulltextSubscriber

Use case: GetFeed
ID: 1
Brief description: The system return a web syndication feed containing information about content that has recently been created or updated
Primary actors: Subscriber
Secondary actors: None
Preconditions: None
Main flow: 1. The use case starts when the Subscriber requests a feed via the network 2. The system verifies that there is content for the feed 3. The system generates the feed using the options given in the request 4. The system returns the generated feed to the Subscriber
Postconditions: None
Alternative flows: GetFeed:MissingFeed

Table 1. Example of a use case specification

inherits all the use cases of Subscriber and adds a third use case, GetFulltextFeed. In order to keep use cases as small and focused as possible, one common area of functionality, LogUsage has been separated out into a separate use case and incorporated via the include mechanism. This use case is more than just an implementation detail: a requirement of the new system identified in discussion with stakeholders is that it should provide more comprehensive logging of requests than the legacy system it replaces, to enable the business to make informed decisions about how to support web syndication in the future (this logging data accessed by the CreateUsageReport use case). We include these use cases on the diagram to reflect their relative importance to the business.

In other cases, arrowheads have been omitted from actor-use-case relationships where they do not provide significant additional value, following [11].

Use case specification In order to describe each use case in more detail, we can use the tabular structure for use case specifications recommended by [12], an example of which for one of the use cases shown in Fig. 2 (GetFeed), can be seen in Table 1. In this use case the primary actor (the one who triggers the use case) is the Subscriber.

Analysis model The above use cases, once identified, can be used to produce an analysis model that focuses on *what* the system does (but not yet *how* it does it). Any artefacts produced in this workflow should remain useful to as

many stakeholders as possible by using language from the business domain and avoiding implementation details.

The analysis classes shown in Fig. 3 have been identified using noun/verb analysis of the project background and use cases above, and are named based on clearly identifiable 'real world' concepts within the problem domain. They are necessarily high-level at this stage; moreover, features that are typically considerations of design rather than analysis, such as operation parameters, visibility adornments and constructors, have been omitted, as per [12].

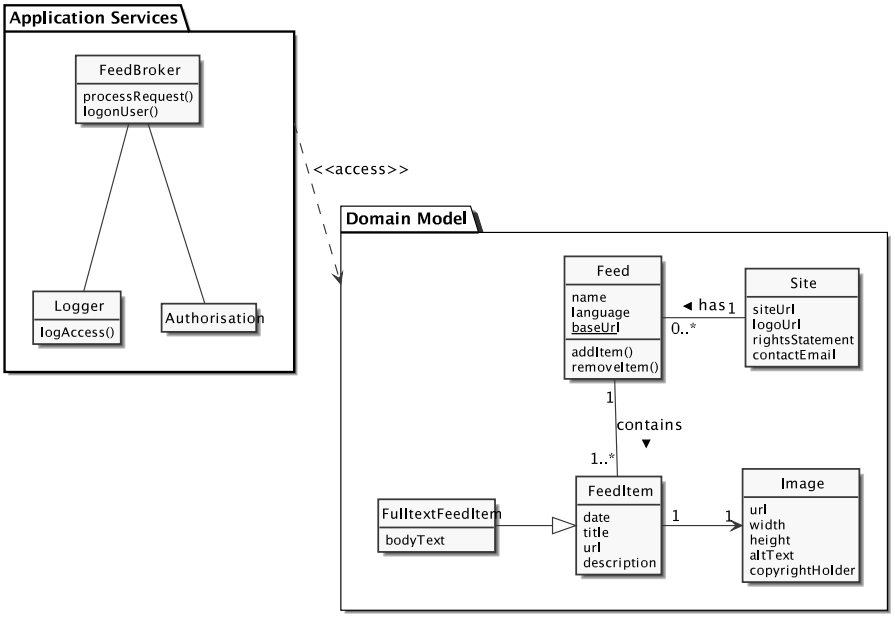

Fig. 3. Analysis classes

The contents of these packages incluce the foundations of a *domain model* [13] for the system that expresses both data and behaviour for the business area, i.e. syndication feeds. However, the only operations shown at this stage within the model (addItem() and removeItem()) are those that may give rise to a state transitions. Also excluded from the diagram are 'getter' and 'setter' methods for each of the attributes. These methods, which are more properly a feature of design classes, could be represented as pairs of operations for each attribute, e.g. getLanguage() and setLanguage(). Ambler [11] considers terms such operations "scaffolding code" and points out that they are often automatically generated by modelling tools. Excluding them from class diagrams, as shown here, here may allow readers to more easily see the most significant operations. This also follows Arlow and Neustadt's [12] recommendation to "only show the classes,

attributes and operations that illustrate the point you are trying to make" in analysis class diagrams.

Use case realisation Use case realisation allows us to investigate how analysis classes collaborate in order to bring about the behaviour specified by a use case. UML sequence diagrams depict a time-ordered series of interations between instances of different classifiers (here, analysis classes). This allows us to show a dynamic view of some of the static class structure seen in the last section.

Fig. 4 shows a sequence diagram which realises the GetFeed use case seen earlier. This type of diagram is a useful way of communicating the flow of messages between classifiers in a particular situation, in order to give the analyst a better understanding of the roles of each participant and to refine the analysis model further if required. As the left-to-right layout of sequence diagrams can be less familiar to non-techical users than traditional flowchart-style diagrams (see later section on activity diagrams), we have used UML notes to provide a 'script' (organised along the left-hand side of the diagram) to allow the reader to interpret the sequence of events more easily.

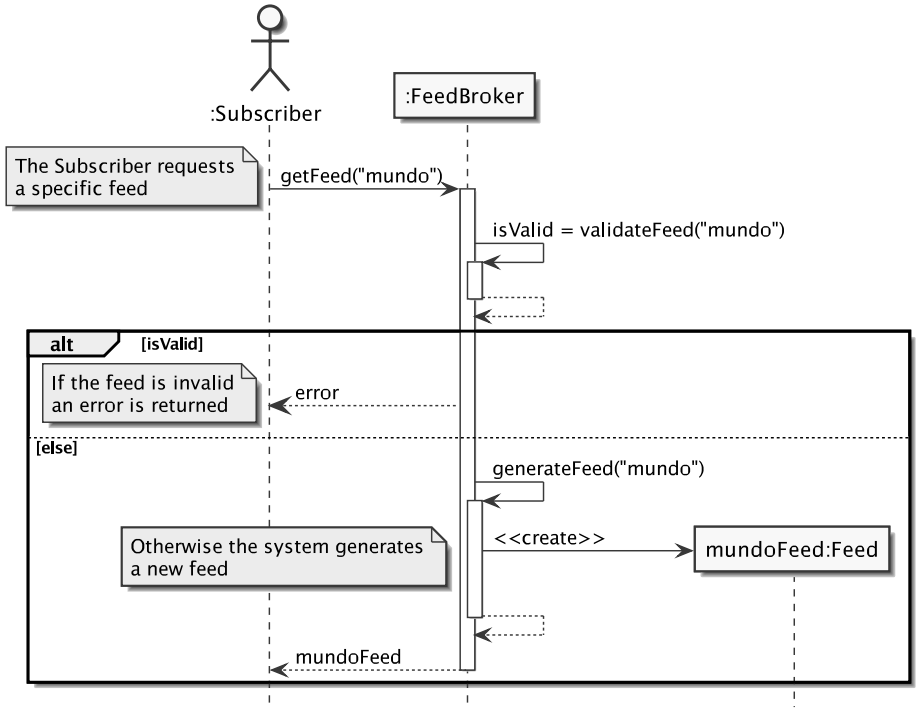

Fig. 4. Sequence diagram for GetFeed

The result of the `validateFeed` self-delegation message sent by FeedBroker is assigned to a temporary variable, isValid. This enables it to be used in the guard conditions of the alt operator and avoids any repetition of the `validateFeed` message within the alt section and therefore confusion about how many times it is called. It also reflects a common convention for implementation at the code level.

Note that the FeedBroker object is unnamed (it does not have an identifier before the colon and class name), whereas the specific Feed instance has been given a name (mundoFeed) This follows the recommendation given by [11] only to name objects in sequence diagrams when they need to be referenced in messages, the downside being that such a diagram shows a mixture of named and anonymous objects. We feel that during analysis the benefits in terms of reducing diagram clutter outweigh any such disadvantages.

Design model In the design workflow we typically refine the relationships between classes to make them explicit. Fig. 5 shows a refinement of some of the design classes from Fig. 3. The figure shows only a subset of the domain model classes given in the analysis model in order to focus on one feature, namely the semantics of aggregation relationships.

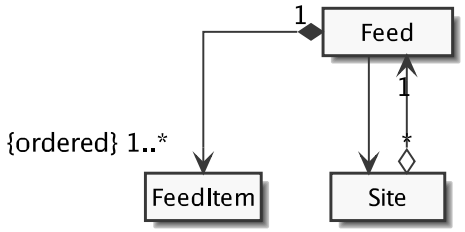

Fig. 5. Refinement of class relationships during design

Note that we have also added the ordered property to the 'part' end of the relationship. This indicates that this one-to-many association must be implemented as an ordered collection of some kind, without placing unnecessary constraints on the implementor by specifying an explicit collection class name. This allows for the most appropriate choice of collection class or type to be used at development time provided it satisfies the constraint (note that FeedItems are also implicitly unique within the collection).

Activity diagram After looking at the functionality the system must provide to satisfy use cases, in the design phase we are concerned with the specific details of the system's implementation. Part of this involves "merging in technical solutions from the solution domain" [12]. One architectural constraint that was placed on this project during design was the choice of application technology

stack. In order to promote cooperation and resource sharing between different organisational divisions, Morph, a new Node.js application platform developed by BBC Sport, is being used to implement feed generation. Morph has been engineered with a focus on efficiency and provides substantial component-level caching via a data flow model.

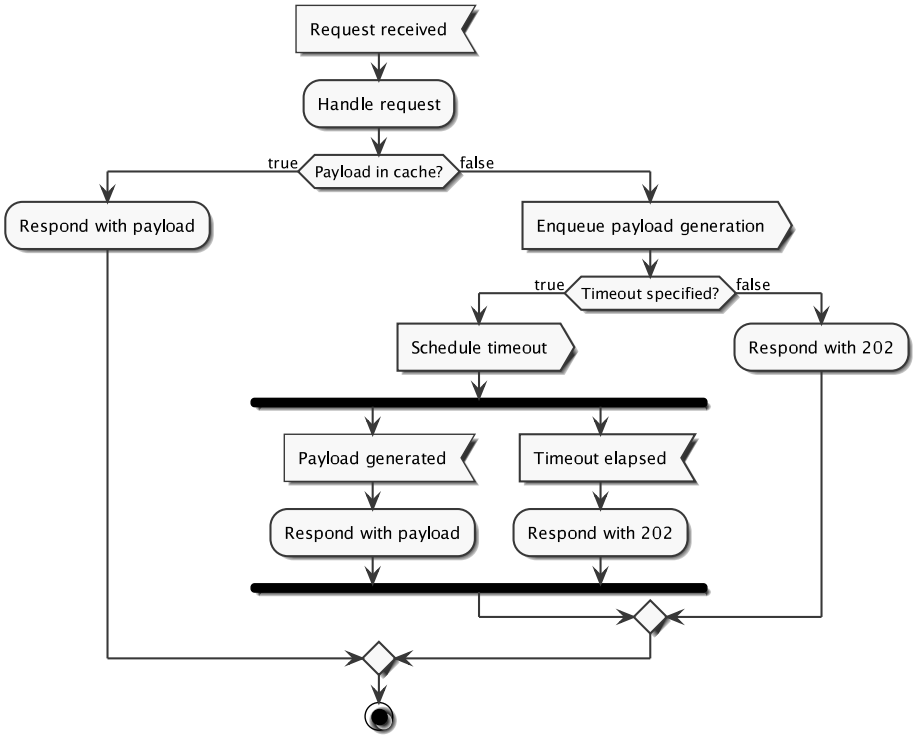

Fig. 6. Activity diagram for Morph caching behaviour

Like the majority of dynamic web applications, Morph uses a pull-based model for generating content, the trigger for resource generation being a request made by an HTTP user agent. Once generated, payloads are cached (using Redis, and optionally, Amazon S3). What is different about Morph is that in the event of a cache miss, by default it immediately returns a `202 Accepted` HTTP response [14] to the user, while the request to generate content is queued for asynchronous generation. The semantics of this response are that the client's request was accepted and scheduled for processing [15].

This corresponds to the 'Fail Fast' design pattern proposed by [16], in that the API responds immediately when it determines that it does not have any representation available. One benefit of this approach is that the client can implement a variety of different strategies for retrying and/or recovering, depending

on the type of data or functionality being provided. A UML activity diagram outlining the above approach is given in Fig. 6.

4 Discussion

Some of the choices made when producing the artefacts in the previous section merit further consideration. For example, the actor names used in the use case diagram (Fig. 2) could be refined, and further roles added, in response to further analysis, since as Arlow and Neustadt [12], stress, use case modelling is an iterative process. This provisionality applies equally to the system boundary (represented by the black rectangle) that indicates the *subject* of the diagram. For example, for the purposes of illustration we have shown the AddContent and UpdateContent use cases as within the system boundary, but in reality these activities are largely decoupled from the syndication feed management process and could be fulfilled by a separate subject. Moreover, as Fowler [17] points out, getting the specific details of actors right is normally less important than finding out the use cases themselves.

The use case specified in more detail in Fig. 2 has one major possible alternative flow, which occurs when the subscriber requests a non-existent feed (MissingFeed). An alternative approach would have been to use the *If* and *Else* keywords within the main flow of Table 1 to describe an alternate flow without creating an entire separate specification. This option was rejected on the grounds that it might hinder intelligibility for non-technical readers, since "use cases are all about communication with the stakeholders" [12].

In the analysis classes outlined in Fig. 3, the items attribute of the Feed class is shown with the multiplicity expression [0..*] to indicate that `null` is a possible value for this attribute, if there are no items. As an alternative to representing class associations diagramatically, we could have shown these relationships using an explicit attribute within the class box (for example, items : FeedItem[0..*]). The diagramatic approach is in our view easier to immediately understand, and avoids having to make assumptions about how associations are implemented during analysis.

To indicate navigability between classes, we followed [12] in suppressing crosses and using a single arrow for unidirectional associations. Following this idiom, it is implicit that the association between Feed and Site, for example, is bidirectional, whereas the association between FeedItem and Image is explicitly one-way. This improves the overall legibility of the diagram, at the expense of some ambiguity.

Although UML supports adding role names at one or both ends of an association to indicate how each class is involved, [11] states that the association name should be sufficient to make roles clear, reserving explicit role names for situations where there are multiple associations between the same classes.

The FulltextFeedItem represents a specialisation of the generic FeedItem which maps to the capabilities required by the GetFulltextFeed use case described previously. In this situation we have decided to inherit all of the existing data and

behaviour of the Feed class and simply add a bodyText member. An alternative approach would be to create an abstract GenericFeedItem class and then provide concrete implementations for both the standard and full-text versions of feed items. However this approach would add greater complexity, and we feel that the risk of coupling FulltextFeedItem to FeedItem is low given there is no requirement to override any of the superclass's operations.

It should also be noted that although the analysis model is intended to represent a generic syndication feed, irrespective of any specific serialisation format, it is somewhat aligned to the structure described by the RSS 2.0 Specification [3]. This is because Atom, the other main competing syndication format, is by and large a functional superset of RSS 2.0, and almost all features of RSS 2.0 can be mapped to an Atom feed without difficulty (for a detailed comparison, see [5]. Moreover, RSS 2.0 is more widely supported and familiar to developers. For this reason we treat it as a "lowest common denominator" of syndication feed formats, while explicitly allowing the system to provide Atom feeds as well.

When considering design classes, attributes and association names have been omitted from Fig. 5 for clarity, in order to clearly show the semantics of the associations. For example, Feed and FeedItem form a whole-part relationship, and because multiplicity on the 'whole' side (Feed) is 1 we can consider this a composition relationship (since a FeedItem may only belong to one Feed and can have no meaningful existence outside of a Feed. Furthermore, feed items share their persistence life cycle with the whole feed [11]. The implication of Fig. 5 is that an instance of Feed can navigate to the FeedItems it contains but not vice versa; if we wanted to allow navigability in the other direction we would have to represent this as an unrefined association arrow in the opposite direction (as seen between Feed and Site) to avoid breaking the asymmetry constraint of aggregation relationships [12].

A more thorough development of the design model is beyond the scope of this work, however such an activity would be expected to cover issues such as member visibility and typing of operation parameters and return values. The limited view here is intended to draw attention to just one of the changes in focus that occur when we move from the analysis to the design stage.

Finally, it is worth mentioning that the activity diagram (Fig. 6) has no explicit initial node, as the *trigger* for this activity is the 'accept event' action node (Request received). This node has been placed at the top center of the diagram to make it more natural to read, following advice by [11]. The signals used in this diagram could be explicitly modelled on a design class diagram using the signal prototype.

As this diagram indicates, the Fail Fast behaviour described above is modified by the provision of a `timeout` parameter. This causes Morph to wait for a defined period of time for the response payload to be generated before returning the 202 response. When this parameter is specified, the outcome depends on which of the two 'accept event' actions are triggered first. An alternative way of representing this would be to use an interruptible activity region in conjunction with an

'accept time event' action node [12], but this more esoteric notation has been avoided here in order to make the diagram simpler to understand.

About the figures The figures in this study were generated using PlantUML, a free open-source tool [18] that generates UML artefacts from plain text files.

The UML 2.0 Superstructure specification [10] states that interaction diagrams (of which sequence diagrams are a specialised form) are surrounded by a "solid-outline rectangle" including the interaction name in the upper left corner. Since at the time of writing PlantUML does not support automatically generating this rectangle, it has been omitted from the diagram included in this document. Interestingly, PlantUML can produce a 'frame' icon that fits the above description when generating deployment diagrams. Since PlantUML is open source software, a helpful contribution to this project (and to the wider UML community) would be to add support for automatically adding the rectangle and title to interaction diagrams in order to make them compliant with the specification.

Note that frames are not missing from the other diagram types such as class diagrams because they are not mandated by the specification for these types. This follows the approach taken in the figures in [12].

5 Conclusions

This report has included examples of only five of the concrete diagrams in the UML specification, and has only considered the first three workflows of the Unified Process. Further studies could be undertaken to asses the usefulness of other diagram types such as communication diagrams and state machine diagrams in developing similar projects. One particularly rich area for investigation could be the relevance of UML component and deployment diagrams in the light of the recent explosion in Cloud computing and the move away from on-premises hardware infrastructure and towards Continuous Delivery (CD).

One challenge of using UML for the system being considered here is produced by the technical requirement of the Morph platform being implemented using Node.js. This entails the use of JavaScript as an implementation language. Although JavaScript has matured somewhat since its early days as a web scripting language and is now widely used for back-end systems, it presents some specific challenges since while UML is strongly aligned with the classical inheritance model used by C# and Java, JavaScript is object-oriented but untyped, (until wider adoption of ECMAScript 6) classless and uses protoypal inheritance. As such, many features of UML class diagrams such as abstract classes and templates are not relevant to JavaScript implementors.

Following on from this, it is important to recognise that while UML offers an effective way of capturing and analysing requirements that describe *what* the system should do, it has limited provision for what are termed "non-functional requirements". Constraints on system design such as choice of programming language, but also broader categories such as performance, security and accessibiity

do not easily map onto the use case-centric view of systems design shown in UML and must be captured separately (this is considered in [12]). It is evident that any user of UML must take special care not to neglect these vital requirements at each stage of the software development life cycle.

It is also worth considering why, given the power of UML both as an analysis tool and as a way of communicating design ideas which we have sought to demonstrate in this study, it is not (at the time of writing) now widely used outside of certain specific industrial contexts. It has been noted [20] that the use of UML may encourage making design decisions in the early pages of the project lifecycle. One explanation therefore for the apparent rejection of UML in organisations such as the BBC is the rush since the early 2000s to embrace apparently lower-risk Agile methodologies and UML's association with a now-unfashionable waterfall model of software development. This aversion to 'big bang' development has been particularly pronounced in the BBC since the failure of its Digital Media Initiative project [19].

That said, the BBC is by no means alone in this regard. In a recent study [21] 70% of industry respondents said that they did not use UML at all, with most of the remainder saying that they only used it 'selectively' (it is worth noting that class, sequence and activity diagrams were among the most used, which supports some of the choices made by this study). One of the main objections cited to UML in the survey was that it focuses too narrowly on architecture and neglects the wider context. This is the antithesis of Checkland's holistic approach to purposeful systems which are always viewed in the context of their environment and the world-view of their users. There is substantial scope for further work on how these two complementary approaches might be usefully combined.

References

1. British Broadcasting Corporation (BBC): BBC announces ambition to double global audience to 500 million, http://www.bbc.co.uk/mediacentre/latestnews/2013/dg-global-audience
2. World Wide Web Consortium (W3C): Channel Definition Format (CDF), https://www.w3.org/TR/NOTE-CDFsubmit.html
3. RSS Advisory Board: RSS 2.0 Specification. Version 2.0.11, http://www.rssboard.org/rss-specification
4. Internet Engineering Task-Force (IETF): The Atom Syndication Format https://tools.ietf.org/html/rfc4287
5. Ruby, S.: Rss20AndAtom10Compared, http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared
6. Hmedeh, Z., Vouzoukidou, N., Travers, N., Christophides, V., Du Mouza, C. and Scholl, M.: Characterizing web syndication behavior and content. In: International Conference on Web Information Systems Engineering (pp. 29-42). Springer, Berlin (2011)
7. Stolz, A., Hepp, M.: From RDF to RSS and Atom: Content Syndication with Linked Data. In: Proceedings of the 24th ACM Conference on Hypertext and Social Media (pp. 236-241). ACM, New York. (2013)

8. Checkland, P., Poulter, J.: Learning for Action: A Short Definitive Account of Soft Systems Methodology and its use for Practitioners, Teachers and Students. John Wiley, Chichester (2006)
9. Checkland, P., Scholes, J.: Soft Systems Methodology in Action. John Wiley, Chichester (1990)
10. Object Management Group (OMG): OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1,
http://www.omg.org/spec/UML/2.4.1/Superstructure
11. Ambler, S. W.: The Elements of UML 2.0 Style. Cambridge University Press, New York (2005)
12. Arlow, J., Neustadt, I.: UML 2 and the Unified Process. Second Edition. Addison-Wesley, Upper Saddle River (2005)
13. Fowler, M., Patterns of Enterprise Application Architecture. Addison-Wesley, Boston (2003)
14. Internet Engineering Task Force (IETF): Hypertext Transfer Protocol – HTTP/1.1,
https://www.ietf.org/rfc/rfc2616.txt
15. Wessels, D.: Web Caching. O'Reilly, Sebastopol (2001)
16. Nygard, M. T.: Release It!: Design and Deploy Production-Ready Software. The Pragmatic Bookshelf, Raleigh (2007)
17. Fowler, M., UML Distilled. Second Edition. Addison-Wesley, Reading (2000)
18. PlantUML.com: Frequently Asked Questions
http://plantuml.com/faq.html
19. British Broadcasting Corporation (BBC): The BBC's management of its Digital Media Initiative,
http://downloads.bbc.co.uk/bbctrust/assets/files/pdf/review_report_research/vfm/digital_media_initiative.pdf
20. Fenning, R., Dogan, H., Phalp, K.: Applicability of SSM and UML for Designing a Search Application for the British Broadcasting Corporation (BBC). In: BPMDS 2014 and EMMSAD 2014, pp. 472-486, Springer, Berlin (2014)
21. Petre, M.: UML in practice. In: 35th International Conference on Software Engineering (ICSE 2013), 18-26 May 2013, San Francisco, CA, USA, pp. 722731 (2013)